Information Processing in Japan Vol. 11, 1971

The Design Language of Computer

HIROSHI HAGIWARA* AND YOSHISUKE KUROZUMI**

1. *Introduction*

The steps of computer design are 1) system design, 2) logical design, 3) packaging design, and 4) circuit design. In this paper, we deal with the computer-aided design language in system design and logic design. Since first proposed by Iverson [1] in 1962, many describing languages or design languages in these fields are introduced and discussed. There are Dully's [2], Friedman's [3], and other people's. We can call these languages register transfer language.

In Japan, Takashima [4] studied the simulation of logical design in 1963. The first design language was proposed by Okada [5] and Motooka. This logic design language had five levels as the language structure. It is useful that manmachine communication can be done between two adjacent levels. The most problem-oriented level in the five levels is the same as register transfer language. The most machine-oriented one is suitable to a logic chart.

We propose the design language of computer which covers system design and logic design. Because system and logic designs are very difficult steps in computer design, we divide this language into three levels as follows: 1) transfer level, 2) block level, and 3) module level.

2. *Function of Computer Design Language*

It is the design to decide specifications of a system (macro specification) fitting for needs, to decide specifications of parts (micro specification) fitting for the macro specification, and to construct the macro specification by the micro specification. In design of a computer, system and logic designs are macro designs, and packaging and circuit designs are micro designs. So, computer design language should have the following functions.

2.1 *Macro Language*

(1) input language

The input language which is the first language in computer-aided design must be able to describe all capabilities of design.

(2) block of system

There are some blocks of construction such as system, unit, and device in

This paper first appeared in Japanese in Joho Shori (the Journal of the Information Processing Society of Japan), Vol. 12, No. 2 (1971), pp. 93-102.

* Faculty of Engineering, Kyoto University.
** Faculty of Science, Kyoto Sangyo University.

a computer. Macro language must be able to decribe these blocks and the linkage between these blocks.

(3) parallel processing and priority processing

Not only parallel processing in each step but also parallel processing or priority processing in the above blocks appears in the design of a computer. It seems to be difficult to describe these processing in languages proposed by other papers.

(4) hardware instead of software

One of the reasons for requirement of the design language of a computer is to replace the part which is processed by software with hardware. So, computer design languages must be able to describe all procedures which general programming languages can describe.

(5) simulation

Simulation is essential to the design. It must be perfect describe the timing for simulation.

2.2 *Micro Language*

(1) output language

The program described by this output language is used as input data to make connection lists, parts lists, and time charts in packaging design and circuit design.

(2) block of circuit

Circuit has some blocks such as element, module, and package. It is important to describe these blocks and the linkage between these blocks.

3. *Transfer Level Language (T-language)*

T-language is a language which describes macro specifications. A program described in T-language is called a procedure. A procedure consists of the following four parts.

(1) declaration part

To design a computer, we need many elements and modules such as registers, counters and adders. We declare these name, kind, size, and response time in this declaration part. There are three declarations: 1) module declaration which declares standard modules, 2) block declaration which declares blocks described in B-language (see 4.), and 3) procedure declaration which declares large units described in T-language. For these declarations, a procedure is able to have the block structure, because there are blocks and procedures in the declaration part.

(2) function part

In the function part, combinational logics among modules are described. For example, three register are named A, B, and C. We describe the circuit that if terminal F is off, then the content of A is transfered to C; if terminal F is on, the content of B is transfered to C. Fig. 1A illustrates the circuit described in the sequence part. This circuit is a sequential circuit and branches to two states which increase the number of states and reponse time. Fig. 1B illustrates the circuit described in the function part. This circuit is a combinational circuit which can select registers in high speed and no branch of state.

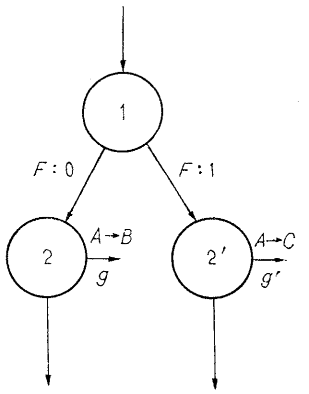


Fig. 1A. Barnch of state in sequential circuit.

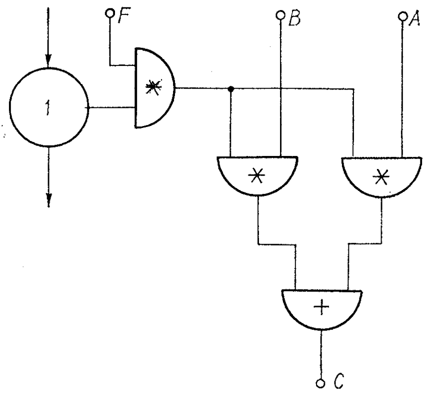


Fig. 1B. Selecting circuit in combinational circuit.

It is useful to be able to describe a combinational circuit simplified instead of sequential circuit. A logical function which appears often in the sequence part must be declared in the function part, and in the sequence part, we can use only function name.

(3) sequence part

Gates among modules and sequences of generating control signals are described in the sequence part. This is a time chart described in a programming language.

(4) control part

In this part, program-control instructions for conversion and simulation are described. As a description language of computer, three parts: (1), (2), and (3) are used. As a design language or simulation language, four parts must be used.

3.1 *Declaration Part*

Modules declarated by a module declarator have standard interfaces and timings. These modules are converted to M-language according to the standard specifications.

(1) **memory**

memory declares not only main memories but buffer momries, general registers and i-o-buffer registers.

ex. 1 **memory** MEMO (CORE(0 : 31), AR(0 : 15),0);

MEMO is a memory module that its capacity is 2^{16} words and its word length is 32 bits. CORE is a data register and AR is a address register. The last 0 in ex. 1 is response time. Zero shows that a memory is asynchronous such as a magnetic drum.

ex. 2 AR=3, DATA=CORE; (Read from the third location.)
AR=5, CORE=DATA; (Write to the fifth location.)

(2) **register**

register declares flipflops or registers consisted of them. A flipflop has many kinds such as RS, JK, T, D, and RST. As the method of designing a gate between these fiipflops is different, the attribute symbol such as *rs*, *jk*, *tg*, *dl*, and *st* is written after a register declarator. If there is no attribute symbol, flipflop is considered to be RS.

ex. 1 **register** DATA (0: 31), ***F***, ***G***, ***W***; **register** *jk* JKFF (0: 5);

ex. 2 ***W***=DATA(0); DATA (0)=DATA (31); DATA (31)=***W***;

(3) **counter**

This is a up or down counter that its step is ±1.

ex. 1 **counter** ***C*** (0: 3);

ex. 2 ***C***=***C***+1; **if** ***C***=7 **go to** ABC;

(4) **adder**

A adder declared by adder has the function that the operands are the outputs of two registers and one flipflop, and a sum or defference of three operands can be sent to the other register.

ex. 1 **adder** ADDER (SUM (0: 31), ***C***, 3);

ex. 2 DATA=***X***−***Y***; DATA=***X***+***Y***+***F***;

(5) **shifter**

shifter declares a shift unit which shifts many bits in a unit time.

ex. 1 **shifter** SHIFT (SHIFTD (0: 31), SHIFTC (1: 4), 4);

ex. 2 DATA=***C*** shr ***X***; ***X***=10 shl ***X***;

(6) **internal**

internal declares internal terminals. It is useful to linkage among modules and redefine names to modules.

ex. 1 **internal** INT (0: 31);

(7) **external**

external declares external terminals. Declarators without **external** are local which effect nothing to the outer programs. But, a external terminal is global which is used to linkage the outer program.

ex. 1 **external** EXT (0: 31);

(8) declaration of multi-dimensional array

About registers and terminals, 2-dimensional array appears in the declaration part. In this case, a cross section of the array is often used by statements of the sequence part. A element of the array is merely used. So, declaration of array in our language is different from it in the standard programming language.

ex. 1 **register** REG (1 : 10) (1 : 10)

ex. 2 REG (1)=REG (2);

3.2 *Function Part*

The function part has only function statements which show the relation of logical function among modules.

ex. 1 **function** $\boldsymbol{A}(0)=\boldsymbol{B}(1)$, $\boldsymbol{F}=\boldsymbol{A}(0)\vee\boldsymbol{A}(1)$;

The following expressions are useful to simplify the long expressions which often appear in practice.

ex. 2 **function** $\boldsymbol{F}=(\vee\boldsymbol{A})$, $\boldsymbol{G}=(\wedge\boldsymbol{A}(\boldsymbol{I}), \boldsymbol{I}=1, 31, 2)$, $(\boldsymbol{A}(\boldsymbol{I})=\boldsymbol{B}(\boldsymbol{I})\wedge\boldsymbol{F}, \boldsymbol{I}=0, 31)$;

The first expression shows logical OR of all elements. The second expression shows logical AND of odd number elements. The third one shows 32 expressions of similar form.

3.3 *Sequence Part*

In the head of the sequence part, subroutines are described. There are open subroutine and close subroutine. These declarators are open and close respectively.

The body of the sequence part consists of the following statements. A separator of statement is ;(semicolon).

(A) Unconditional statement

A unconditional statement explains some actions in the state. It has some instructions separated by ,(comma). The expression that a statement consists of some instructions simplifys the description of parallel processing.

(1) assignment statement

A assignment statement has the following instructions.

simple assignment instruction	ex. $\boldsymbol{A}=\boldsymbol{B}$,
arithmetic assignment instruction	ex. ACC$=\boldsymbol{A}+\boldsymbol{B}$,
logical assignment instruction	ex. $\boldsymbol{F}=\boldsymbol{A}(0)\vee\boldsymbol{A}(1)$,
shift assignment instruction	ex. $\boldsymbol{B}=\boldsymbol{C}$ **shr** $\boldsymbol{B}$,

(2) call statement

This statement has at least one call instruction. A call instruction calls a open or close subroutine.

ex. SUB; OPEN (C1, C2);

(3) **go to** statement

A **go to** statement is used to indicate the junction of sequence. A label is written after **go to.**

ex. **go to** ABC;

(4) compound statement

To describe such parallel process that its processing time is variable, a compound statement is used. A compound statement consists of some compound instructions.

ex. **begin** S; S; $\cdots$ S **end**, **begin** S; S; $\cdots$ S **end**;

(B) Conditional statement

This statement shows a branch of state and does not generate new states.

(1) **if** statement

A **if** statement is used to branch to two states or wait in a state.

ex. **if** ***F*** **then** go to ABC; if ***F*** **then** wait;

(2) **switch** statement

This statement is used to branch to many states.

ex. **switch** INS (0:3)=S0, S1, S2, ···, S15;

(C) Label

Labels are able to be added to the head of a statement. A label shows the joinning point of sequence and the phase of a state.

ex. ABC: ***A***=***B***; ABC*1: ACC=***A***+***B***;

3.4 *Control Part*

A conversion method of control circuits is designated in this part. The conversion methods are sequence, assignment, and microprogram.

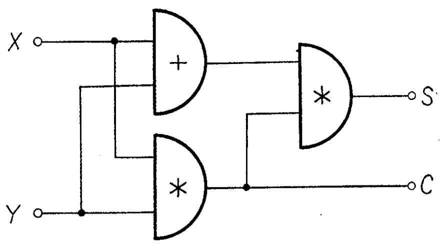
ex. **control assignment jk** (*L*1, *L*2, ···, *L*10);

And, the control of simulation is designated in this part. These designator are input, output and time.

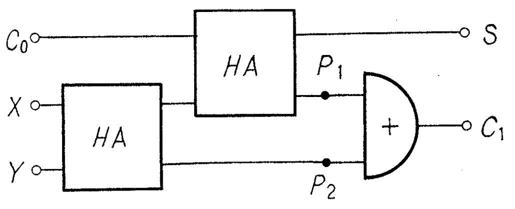
4. *Block Level Language (B-language)*

B-language is a language which describes micro specification. A program described in B-language is called a block. A block consists of module declaration, block declaration, function, and block call. The module declaration and function are explained in T-language. A block call is the same as subroutine call statements. It calls a block which declared by a block declaration.

Example A	Example B
block $HA(X, Y/S, C)$;	block $FA(X, Y, C0/S, C1)$;
$S=(X\vee Y)\wedge\sim C$;	internal $P1, P2$;
$C=X\wedge Y$;	block $HA(XH, YH/SH, CH)$;
end;	$SH=(XH\vee YH)\wedge\sim CH$;
	$CH=XH\wedge YH$;
	end;
	$HA(C0, P1/S, P2)$; $HA(X, Y/P1, P2)$;



A: Half adder



B: Full adder

Fig. 2.

$C1=P1\vee P2$;

end;

5. *Module Level Language (M-language)*

This language is suitable to describe the connection tables. It consists of only module statements as follows.

⟨module statement⟩: : =⟨label⟩ : ⟨module name⟩
(⟨input terminal list⟩/⟨output terminal list⟩)

ex. BOOL: BOOL-1AN(*A*(0)-RS1, *A*(1)-RS1/BOOL-3OR)

6. *Conclusion*

We propose the design language of a computer. But it is important that we describe many computers in this language and we study to make a converter and a simulator of this language.

References

[1] Iverson, K.E.: A programming language. John Wiley and Sons, NewYork (1962).
[2] Dully, J.R. and D.L. Dietmeyer: A digital system design language (DDL). *IEEE Trans, C-17*, pp. 850-860 (Sep. 1968).
[3] Friedman, T.D. and S.C. Yang: Methods used in an automatic logic design generator (ALERT). *IEEE Trans, C-18*, pp. 593-614 (July. 1969).
[4] Takashima, T. et al.: A universal program system simulating logic. *Information Processing of Japan*, pp. 64-72 (March. 1963).
[5] Okada, H. and T. Motooka: Logical design language. *The Journal of the Institute of Electronics and Communication Engineers of Japan*. pp. 2353-2360 (Dec. 1967).